Tech Basics II:
Hangman, but cooler

Major Digital Media

Louis Rohrbach
louis.rohrbach@stud.leuphana.de
Leuphana University Lüneburg
Matriculation Number: 3037215

Course: Tech Basics II Stream A
WiSe 2019-20
Helena Lingor

Table of Contents

1. Introduction

1.1 Motivation

This project is the continuation of my first project from the course Tech Basics I. Since my first iteration of the game was built solely around the terminal, I wanted to implement the required use of a Graphical User Interface into the game. I still think the game is fun and challenging, especially when the word list is generated. The first project did not have any graphical interface outside of the terminal and it wasn't as visually pleasing as the TKinter GUI. It essentially is an upgraded version of the traditional game that can be played with two players, but with a computer as a counterpart, that chooses random words out of a word list.

2. Main Structure and Functionalities

The main overall structure stayed the same: a .txt file with a list of capitalized words is given, the program chooses a word out of that list and transforms it into a secret word, which is only shown as underscores. This, as in the terminal application used in the Tech Basics I project, word is word that has to be guessed by the user, letter by letter. The user has 9 tries to guess the letters contained in the secret word. If the guesses are used up, the user loses and the word is revealed.

In last semesters application, the wrongly guessed tries were pictured by a hangman drawn by underscores and slashes. This project however, utilizes the Tkinter library, which allows the game to be played utilizing a mouse and the graphical interface instead of the keyboard and the terminal.

Modules like “import random" and “from tkinter import *” are necessary for the application to work. This time, the code begins with creating the TKinter root window, which is essentially the GUI that will be used throughout most of the game. It gains a

title, and the attribute of being non-resizable, to provide consistency when playing the game. Since the game uses png files for the showcase of the hangman instead of self-made drawings in the terminal, the images have to be loaded in the backend of the application using the command “ImageTk.PhotoImage(Image.open())”. The program uses a total of ten images, ranging from an empty white image to the fully drawn out hangman. These images are brought together in a list “image_list” to be easily accessible and interchangeable throughout the game. As soon as the user guesses a wrong input, the item used from the list increases by one to show the next picture.

The GUI elements that are shown are the hangman image on the left, the frame that is used by the underscores and letters for the words and the keyboard. Every key of the keyboard is placed according to a real QWERTZ keyboard and mapped to its letter accordingly, with a “Quit Game” button implemented in the bottom right corner that quits the whole game with “root.quit()”. All the pressable buttons are initialized at the end of the code before the main function.

The initial function used is “start_game”, which randomly chooses the word from the secret_word.txt file with “random.choice(secret_word)”, replaces the letters with an underscore and joins a space to look more aesthetically pleasing. This is displayed with the help of a StringVar, a string variable that can be changed throughout the application. This is saved as “frameWord” and displayed in the label at the top of the GUI. This label has the “columnspan=10”, to span across the keyboard in Times New Roman and font size 50. The game utilizes this StringVar to display correctly guessed letters instead of underscores. This function also saves the secret word as the variable “word_with_spaces” for later usage.

Another function that only serves a purpose when repeating the game is the “pic_reset” function, which forgets the current shown hangman image, loads the first blank image into the backend and places that image. This function is used when certain scenarios appear, that prompt a possible reset. This is used when the end message, which will be explained later, appears and the reset prompt is activated.

At last, the “user_guess” function is the most important. This function is defined as the basic ruleset of the game. “start_game” initializes the word and places it into the label, “pic_reset” is only used in restart-scenarios, and every other GUI element is booted before running the main loop of the game. Every keyboard button, except “Quit Game” is programmed to communicate with the “user_guess” function, using a lambda prompt when pressed. Pressing the letter A for example prompts the “user_guess” function for the letter A.
Previously mentioned variable “word_with_spaces” is transformed into a list, when the function is called. This list “letter_list” is used for comparison. On the other hand, the word that is shown in the frameWord, which starts off as a blank word only shown with underscores, is transformed into “secret_letter_list”. Both of these list contain singular letters as their elements, which can be easily compared. Every time this “user_guess” function is called, the word that is displayed is transformed and compared to the list of letters of the word that is saved under “word_with_spaces”. The next part is the most crucial part, where the the letter, if guessed correctly, replaces the underscore that represented it in the label “frameWord”. It checks for the length of the list of the list of “word_with_spaces” and prompts an if function, that uses the user input to replace the correctly guessed letter with “frameWord.set(“”.join(secret_letter_list))”.
If the letter is guessed incorrectly, an else function triggers which raises the number of the hangman picture list that was mentioned earlier. When advanced, a variable called “picture” increases by one and triggers a message box named “restart” when nine is reached with “if picture == 9:”. This means the user has lost.
Another if function that compares both the shown frameWord on the GUI and the previously saved “word_with_spaces” triggers when both are identical. This prompts the “restart2” scenario, another message box. This means the user has won.

Both of the message boxes that are practically the same, showing the word whether the user has won or not. It shows a winning or losing message, together with the secret word and yes or no buttons asking for a restart. If yes is pressed, a new word is taken

from the list, and replaces the currently shown one, while also initializing the “pic_reset” function. If no is chosen, the game quits.

2.1. Progress and Challenges

The progress this version presents in comparison to the prototype from previous semester is tremendous. The most important part is, that the game runs smoothly and without exploits like the last one. This is due to the fact that I changed the whole game ruleset, on how the correct guesses are counted. In the previous version, the game counted the amount of correctly guessed letters, meaning that it checked the guessed letter with the word, and if the letter appeared twice in the word, it would count as two correct letters. However, this presented a bug, which led to the same guess being able to be used multiple times. If the word was “JAZZ”, a guess of the letter Z twice would end the game. This made me realize that I couldn’t use the same game mechanic, vastly due to the major bug, when confronted with the TKinter GUI. This required me to completely change my system of counting the letters to win the game.

Replacing the word with spaces and underscores was directly used from the first project, but the challenge this time was to translate that onto the GUI. The StringVar together with the replacing mechanism was found in a tutorial video[1]. The mechanic, as previously explained creates two lists which are compared with each other. If an element of the list which contains the word is found in the guess, the guessed letter replaces the according underscore that is displayed with the help of the StringVar. This was the most difficult to figure out, since it required a complete restructuring.

The new guessing mechanic allowed me to also change the winning function. By just comparing what is shown in the StringVar with the word that is saved as the secret word in the backend, it was way easier to end the game with that. Instead of counting the letters it was a simple if-function that compared the two. I chose this because this

[1] https://www.youtube.com/watch?v=esHcFbNlNqQ&t=405s

eliminated the bug I had previously while also being easier to use for the winning function. The wrong guesses were counted the same way and only progressed the pictures instead of drawing them in the terminal.

One thing I encountered was that the last letter or the last picture, depending on a win or loss, would not show up before, but after the message box appeared. I tested this multiple times. Even though the message box prompt is called for when the shown StringVar word is the same as the secret word in the backend, it still didn't replace the last underscore or picture before the message box. This is why I chose to opt for the reveal of the word in the message box, so that it still is fully shown at the end.

Programming each keyboard button was time consuming, since it required me to find out the correct position of each key within the columns and rows. This was a long trial and error process, since the columns and rows alter when other elements like the frame for the frameWord or the picture of the hangman are implemented. I also wanted a recognizable interface, which influenced my decision to go with a QWERTZ keyboard instead of keys in alphabetical order. Another key that I didn't have was a simple "Quit Game" key which makes ending a game much easier.

The progress made from the first prototype is big, because it introduced a clean GUI with different mechanics for backend processing. The list for the secret words is the same. The new version doesn't require terminal input, pressing enter after each letter and scrolling through a terminal, but rather a easy-to-understand GUI with images for better understanding of the game.It also reveals the word in a better way while granting the option to quit or restart after playing it.

3. Bibliography, Figures and Necessary Data

The module “import random” is used to utilize the function “random.choice()”
The tkinter library was used and the module PIL was utilized.

1 Code for the the two lists for replacing letters was found at https://www.youtube.com/watch?v=esHcFbNlNqQ&t=405s

The .txt file database that was used to import the random word can be modified freely. The one used for testing the game contained:

```
GIRAFFE
NECK
MONKEY
TABLE
ZEBRA
HAIR
OCTOPUS
ANNIHILATION
JUDGEMENT
BEER
MICROFIBER
ACRYLIC
LAPTOP
ASSESSMENT
CODE
TROMBONE
MASSAGE
NINTENDO
```

It is important for the code to work, that the letters are written beneath each other in uppercase letters and that the file is named secret_words.txt

The images of the hangman are drawn by myself.

4. Statutory Decalaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Louis Rohrbach